**Proyecto Algoritmos y Computación**

**Análisis de sentimiento para inversión en bolsa**



**Victor Vaquero Martinez**

**Juan Rodríguez Sanz**

**Índice**

Introducción

Inspiración

Objetivo

Problemática

Componentes del algoritmo

Requerimiento Temporal

Etapas:

* 1º Etapa: Extracción de Información
* 2º Etapa: Preprocesado del vector de entrada:
* 3º Etapa: Reducción de Dimensionalidad
* 4º Etapa: Predicción del Sentimiento
* 5º Etapa: Estrategia de Inversión

Bases de Datos

Bibliografía

**Introducción**

En el mundo actual existe una amplia competición en el ámbito inversor para obtener estrategias capaces de extraer décimas de puntos básicos sobre el resto de los competidores. El análisis de la opinión pública es un tema de vanguardia sin aún un gran desarrollo que realizado con éxito podría otorgar a sus creadores ventajas sustanciales sobre la ingente masa de actores actualmente invirtiendo en el mercado de valores.

**Inspiración**

El origen de la idea parte de diversos papeles y blogs sobre el tema [1] [7] además del previo conocimiento sobre la existencia de herramientas para el análisis de este, algunas por ej basadas en análisis de palabras clave [8]. La mayoría de las fuentes usan librerías de big data como TensorFlow [10] con Keras o Pytorch [11] para aplicar varios algoritmos de procesado de lenguaje natural. Es por esto que sabemos que la idea, aunque hasta ahora no realizada al menos de manera pública, es realizable.

Existen multitud de precedentes en el uso de datos externos a los precios de mercado para la predicción de estos, siendo ejemplar el de la empresa Renaissance Technologies [9] creada por el famoso matemático Jim Simmons de la cual se sabe que aplica modelos predictivos basados en la recogida de datos de multitud de orígenes para obtener una ventaja sobre el resto de los inversores [12]. Por supuesto la implementación práctica de dichos modelos se guarda bajo llave para evitar perder dicha ventaja pero es posible inspirarse en estos gigantes de la industria e intentar imitarles, aunque sea con gravosa dificultad.

Por todo esto y dados los recientes avances del procesado de lenguaje natural [6] creemos que es verdaderamente posible llevar a buen cauce este proyecto hasta el final.

**Objetivo**

El presente proyecto tiene como objetivo la simulación de una estrategia de inversión en una bolsa de valores, sobre varias empresas previamente seleccionadas, a través del análisis de la opinión pública junto con el histórico de dichas empresas.

Nuestra hipótesis inicial sobre la que se basa el proyecto es que existe información sobre la tendencia del valor de las acciones oculta en la opinión pública y que de ser analizada puede otorgar una ventaja sobre el resto de los actores del mercado.

En concreto nos vamos a centrar en la información proporcionada por las empresas de Microsoft, Apple, Google, IBM y nvidia a traves de sus cuentas de twitter y de su histórico de precios.

Concretamos dicho valor de la información pública tratando de detectar patrones subyacentes en los movimientos del mercado relacionados con el sentimiento positivo y negativo sobre una empresa objetivo, definiendo este como una media diaria de la totalidad de sentimientos individuales.

Por último utilizando el sentimiento obtenido y el histórico de precios, previamente recogido, hacemos una predicción sobre el posible movimiento de la bolsa en esa empresa y con una estrategia de inversión simple tomamos la decisión de si invertimos o no.

**Problemática**

Previo desarrollo del algoritmo cabe mencionar una predicción sobre cuáles serán las mayores dificultades por las que se pasara hasta la finalización del prototipo.

En primer lugar el análisis de lenguaje natural es un clásico de la inteligencia artificial sobre el cual se siguen desarrollando actualmente soluciones a problemas de hace varias décadas. En particular nuestra decisión de analizar texto no regulado, corregido o sobre todo de tipo más formal se habrá de probar como el mayor muro a la realización del proyecto.

No solo eso sino que dada la subjetiva naturaleza de la opinión popular nos habremos de limitar a un pequeño subgrupo de emociones estimadas, y además será necesario realizar una búsqueda intensiva de las bases de datos etiquetadas hasta encontrar si cabe una decente muestra de datos representativos de lo que deseamos estimar.

Por último es posible que la hipótesis inicial sobre la que basamos nuestro trabajo, que la opinión del público verdaderamente gobierna aunque sea una pequeña parte la realización de los precios del mercado, podría probarse errónea o ser de efecto lo suficientemente pequeño como para hacer impráctico su aprovechamiento.

**Componentes del algoritmo**

En total consta de cuatro partes diferenciadas. Primero un conjunto de bases de datos obtenidas de diferentes páginas web como a través de una extracción de datos utilizando la técnica de web scraping.

Segundo dos redes neuronales en tándem, primero un autoencoder para mejorar nuestra representación del vocabulario a través de una codificación más eficiente y comprimida, y después una RNN ( Red Neuronal Recurrente), en particular una LSMN ( Red de corta-larga memoria), para la predicción de los sentimientos.

Tercero un sistema de modelado de series temporales a través de un suavizado exponencial simple donde incluimos tanto el histórico como la predicción de sentimientos diaria.

Por último, usamos dicho modelo para predecir valores futuros de las acciones y a través de una estrategia de inversión intentamos adelantarnos al mercado.

**Requerimiento Temporal**

Hemos analizado los requerimientos de tiempo y creemos que la etapa más compleja y que requerirá más tiempo es la de preprocesado, pues existen una gran cantidad de variables en el idioma que requieren un análisis muy cuidadoso. Por esta razón decidimos limitar la información extraída a un único idioma, el inglés.

En total dedicaremos una semana por etapa, acabando a tiempo en 6 semanas, pero teniendo en cuenta lo previamente dicho la 2 etapa (preprocesado) nos ocupará al menos dos o más por lo que ha ser posible habremos de acelerar el resto de las etapas.

**Etapas**

**1º Etapa: Extracción de Información**

**Diseño:**

Comenzamos programando varios scripts para extraer información de las diferentes fuentes. En total extraemos los siguientes datos:

Twitter:

*Tweets con el hashtag de cada empresa elegida.*

*Se usarán para realizar la predicción diaria de la opinión pública sobre una empresa.*

* #Microsoft
* #Apple
* #Google
* #IBM
* #nvidia

Yahoo finances:

*Histórico de precios.*

*Se usarán para, junto con la opinión diaria sobre la empresa, predecir los movimientos de precio a futuro. Además servirán para analizar la capacidad de nuestro algoritmo.*

* MSFT
* GOOGL
* IBM
* NVDA
* APPL

Kaggle:

*Base de datos con tweets etiquetados.*

*Se usará para entrenar nuestra red neuronal recurrente de predicción de sentimientos.*

Páginas web:

*Fuentes de confianza de alta relevancia financiera.*

*Se usarán para limitar el vocabulario usado por nuestras redes neuronales, de otra manera sería de excesivo tamaño.*

* Market Watch
* Seeking Alpha
* Barrons

**Desarrollo:**

La extracción de yahoo finance es inmediata pues nos ofrece la posibilidad de descargar el histórico de precios en formato csv. Lo mismo sucede con la base de datos de Kaggle.

Problema:

* Como vamos a necesitar varias bases de datos de distintas fuentes hay que personalizar el script para cada fuente, pues distintas fuentes tienen estructura diferente y por ende etiquetas html diferentes.

Soluciones:

* Utilizamos scripts en python para la extracción de información a traves de twitter puesto que al ser la misma pagina no cambiaran las etiquetas que queremos extraer en nuestro caso los tweets.
* Utilizamos una herramienta externa para la extracción de las páginas web de confianza (Web Scraper una extensión de google chrome).

En ambos casos la extracción proporciona un archivo con extensión csv con cada empresa o fuente de información.

Problemas:

* Twitter nos proporciona un api para la obtención de datos, pero este está limitado a la recogida de datos 7 días atrás.
* De Twitter se obtiene una gran cantidad de tweets por día por lo que para su posterior análisis es excesivo. Aunque es posible su filtración esta está limitada al igual que los tweets (solo te devuelven los más importantes sin especificar cómo realiza dicha distinción).

Soluciones:

* Obtención de datos de twitter a través de api de terceros “GetOldTweets” para python, un web scraper que permite saltarse la limitación de recogida de 7 días de la api oficial de twitter.
* Limitamos a 100 tweets por dia por empresa además de recoger los más “relevantes”, a saber, aquellos con mayor número de retweets, comentarios y megustas.

Resultados obtenidos:

Para cada empresa:

* Opinión pública sacada de twitter.
* Histórico de precios diario.

Para entrenar las redes neuronales:

* Tweets etiquetados.
* Fuentes de confianza con el vocabulario.

**2º Etapa: Preprocesado del vector de entrada**

**Diseño:**

Una vez extraída la información, hay que tratarla previamente procesandola y transformándola a un mismo formato. Para ello, cada fichero primero pasa por un preprocesado que simplifica y normaliza el texto:

* Elimina signos de puntuación, artículos, así como caracteres especiales (como #, @, $, €, entre otros).
* Cambiar todo a minúsculas.
* Cambiar cualquier verbo en cualquier forma verbal a su forma infinitivo.
* Cambiar todas las palabras de plural a singular.
* Se limite a un vocabulario relevante (Ej: las 1000 palabras más frecuentes).

Ejemplo de tweet de entrada al preprocesamiento:

$TSLA Hello world? #TeslaMotors #TeslaModel3 Those last 3 monthly candles are nasty.

hello world those last 3 monthly candle be nasty

Al finalizar el preprocesado obtendremos una lista de palabras completamente normalizadas.

* Lista de palabras de las fuentes de confianza.
* Lista de palabras de los tweets etiquetados.
* Lista de palabras de los tweets de cada empresa.

**Desarrollo:**

**1º versión:**

Problema:

* Para pasar los verbos a su forma infinitivo hay que detectar que esa palabra es un verbo.

Solución:

* Extraer una lista de verbos con todas sus formas verbales, teniendo en la primera columna la forma en infinitivo de ese verbo, y comprobar si la palabra está en esa lista. Si lo está es un verbo con su infinitivo en la primera columna. Si no está entonces esa palabra no es un verbo.

Para hacer la lista hemos recogido los verbos a través de la wikipedia y guardandola en un fichero csv.

Problemas:

* Para pasar las palabras de su forma plural a singular hay que implementar las reglas de transformación de un singular a plural pero de forma inversa:
  + Estas reglas tienen excepciones.
  + Existen plurales irregulares que son totalmente ajenos a estas reglas.
  + Detectar una palabra en plural y no confundirla con una palabra en singular.
  + Existen ambivalencias, palabras que pueden actuar como nombres o verbos cambiando las reglas que se les han de aplicar.

Las soluciones a estos problemas no son triviales por lo que hemos decidido implementar una librería ya creada que nos permitirá la resolución de estos problemas.

**2ª versión:**

La librería se llama “Stanford Parser” que permite, pasando un texto por entrada, devolverte una estructura en forma de árbol con el tipo de cada palabra (nombre, verbo, nombre plural, etc).

Teniendo este aspecto:

(ROOT (S (NP (ADJP (RB often) (JJ capitulation)) (NNS lows)) (VP (VB be) (VP (VBN retested))) (. .)))

Solo nos interesa el tag entre paréntesis y la palabra que lo sigue → (NNS lows). El tag hace referencia al tipo de palabra que es y la palabra siguiente es la palabra que queremos tratar.

Como se puede apreciar podemos distinguir fácilmente cada palabra reduciendo así la búsqueda de verbos, nombres a solo buscar por el tag.

Problemas:

* A partir de cierto número de caracteres seguidos comienza a funcionar mal (textos muy largos sin muchos puntos).
* Tenemos el tipo de palabra pero seguimos sin poder pasar de plural a singular.

Soluciones:

* Si la cantidad de caracteres supera una cantidad colocar un punto en la mitad del texto.
* Utilizamos un scrape de la pagina https://en.wiktionary.org/wiki/nombreDeLaPalabra. Así obtenemos el singular de una palabra en plural o el infinitivo de un verbo.

Por último implementamos una tabla que recoja todas las palabras procesadas y las ordene por cantidad de aparición.

**3ª versión:**

Con la segunda versión ya es posible continuar con el proyecto pero:

Problemas:

* La obtención del árbol de cada texto pasado es muy lenta.
* Hay una gran cantidad de textos a pasar en otras palabras un gran volumen de datos.

Solución:

* Multiprocesamiento (hilos).

La tercera versión solo se ha centrado en mejorar la eficiencia del preprocesado habiendo reducido hasta en un 40% el tiempo con respecto a la versión 2. Esto a su vez genera problemas porque estamos limitamos a la capacidad computacional de nuestros equipos por lo que lo limitamos a una cantidad fija de hilos simultáneos. Ocurre lo mismo para el acceso a la wikipedia, lo limitamos para evitar hacer demasiadas peticiones desde la misma ip en un muy corto periodo de tiempo.

Una vez obtenido el vocabulario de palabras relevantes para el trading de tamaño N, podemos representar cada palabra de nuestra base de datos (de tweets) como un vector N dimensional. Cada tweet estaría representado como un vector N\*M con M número de palabras.

Ejemplo:

Tweet → hello world, those last days...

Tweet procesado → hello world that last day

Vocabulario de los tweets → [hello, that, candle, nasty, be, world]

[1, 0, 0, 0, 0, 0] [0, 0, 0, 0, 0 , 1] [0, 1, 0, 0, 0, 0] [0, 0, 0, 0, 0, 0] [0, 0, 0, 0, 0, 0]

Como los últimos son un vector de ceros, o lo que es lo mismo si no está incluido en nuestro vocabulario, no lo consideramos relevante y puede ser eliminado. Esta representación se llama “One-hot”.

**3º Etapa: Reducción de Dimensionalidad**

**Diseño:**

Dado que este vector completo tendría una dimensionalidad intratable pues tendríamos un vocabulario muy grande y los vectores serian de gran tamaño (EJ: Nº de palabras del vocabulario: 500, Nº de palabras del tweet(aproximado): 100 = 50.000 posiciones ), es necesario realizar esta reducción.

Esto se lleva a cabo a través de una primera red neuronal, un autoencoder basado en el método word2vec [2].

Este algoritmo permite crear una representación más compacta de un vector en el cual palabras relacionadas estén más cerca en dicho espacio reducido.

Se trata de predecir en un conjunto de textos, a partir de una palabra las palabras de alrededor (el contexto). Para ello la red tendrá una entrada de tamaño N\*M, con una capa oculta de reducida dimensionalidad de tamaño E (Ej: 16,32,...) y una salida de dimensión N\*M.

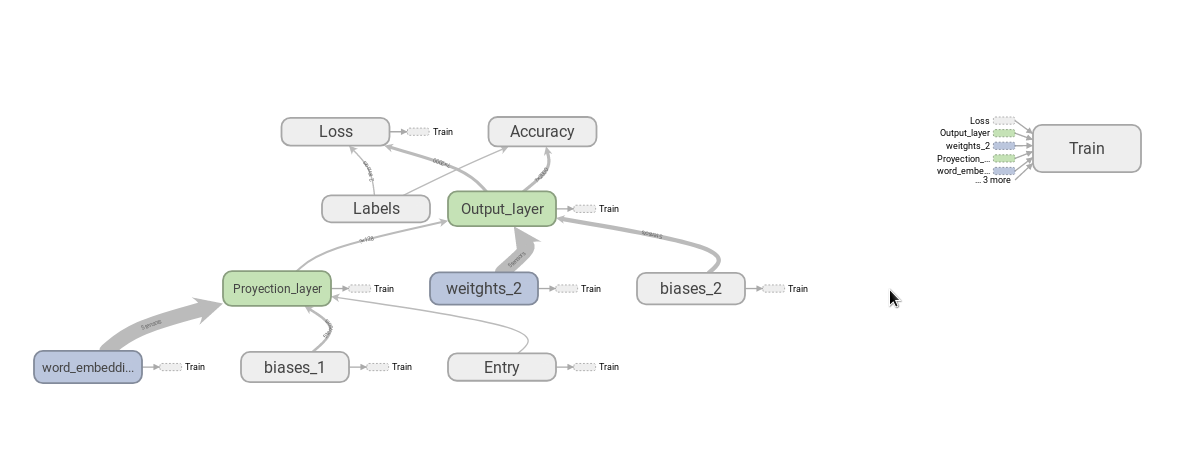
Una vez entrenada la red con ambas bases de datos (previamente preprocesadas), la codificación final se encuentra en dicha capa oculta (los pesos de las neuronas). Nuestra hipótesis considera (esto a sido confirmado en la práctica) que si la red es capaz de predecir el contexto, ha de haber sido capaz de condensar la información en un tamaño de vector E.

Ejemplo: Con una reducción a un espacio de tamaño 3 resultaría en

[1, 0, 0, 0, 0, 0] [0, 0, 0, 0, 0 , 1] [ 0, 1, 0, 0, 0, 0]

→ [ 0.134, 0.875, 0.23] [ 0.87, 0.1, 0.02][0.34, 0.452, 0.1]

Como se puede apreciar se ha reducido a un 50% el tamaño de los vectores.



*Esquema general de la red neuronal, entrada “Entry”, salida “Output\_layer” y funciones de entrenamiento y diagnostico “Loss” y “Accuracy” - Obtenido con TensorBoard*

**Desarrollo:**

Hemos decidido realizar ambas redes a través de la herramienta TensorFlow [10] de Google pues nos facilita la implementación de una red neuronal con la flexibilidad de tener total control sobre su estructura.

Problema:

* En un principio recreamos la codificación inicial de palabras a números en cada etapa ya que variaba continuamente en la etapa de pruebas, craso error pues esto hacía que variará ligeramente qué número era asignado a cada palabra de nuestro vocabulario sobre todo las que aparecen con menor frecuencia.

Solución:

* Es necesario crear un archivo con la relación de palabras y el entero que tienen asignado (implícito por el orden en el que están escritas).

Tras varias iteraciones de prototipos nos establecemos con una reducción a una codificación en un vector de tamaño 64 la cual es la que nos otorga unos mejores resultados. Además tamaños mayores sobrepasan nuestra capacidad computacional.

El primer paso es la creación de tuplas entrada-salida a partir del texto procesado para el entrenamiento.

Problema:

* En la etapa de diseño inicial no tuvimos en cuenta que existen diferentes versiones del sistema word2vec y que este se puede implementar de varias maneras cada una con sus pros y sus contras y con diferentes tuplas entrada-salida.

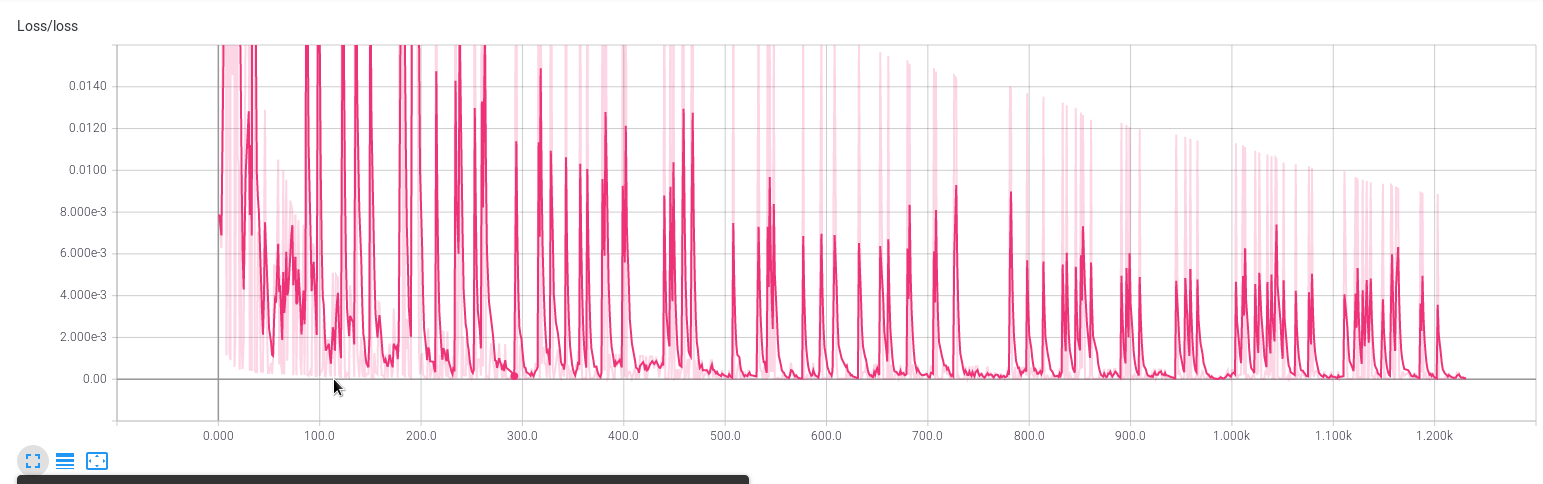
Solución:

* Este caso es sencillo pues tan solo necesitamos refinar nuestra elección previa, en este caso elegimos un modelo “bag of words” en contra del modelo “skip gram” pues este primero posee una ventaja de precisión para casos como el nuestro en el que se posee un número limitado de texto de entrenamiento.

El siguiente paso es la construcción de nuestra red. Se crea el grafo computacional de TensorFlow con las operaciones *tf.multiply* para multiplicar matrices, *suma* para añadir el bias, etc. En total quedan 3 capas, la de entrada en la cual se transforma la entrada a vectores con representación “One-hot” como mencionamos previamente, la capa de proyección que comprime la información y la capa de salida donde se obtiene un vector de probabilidades de cada palabra en V.

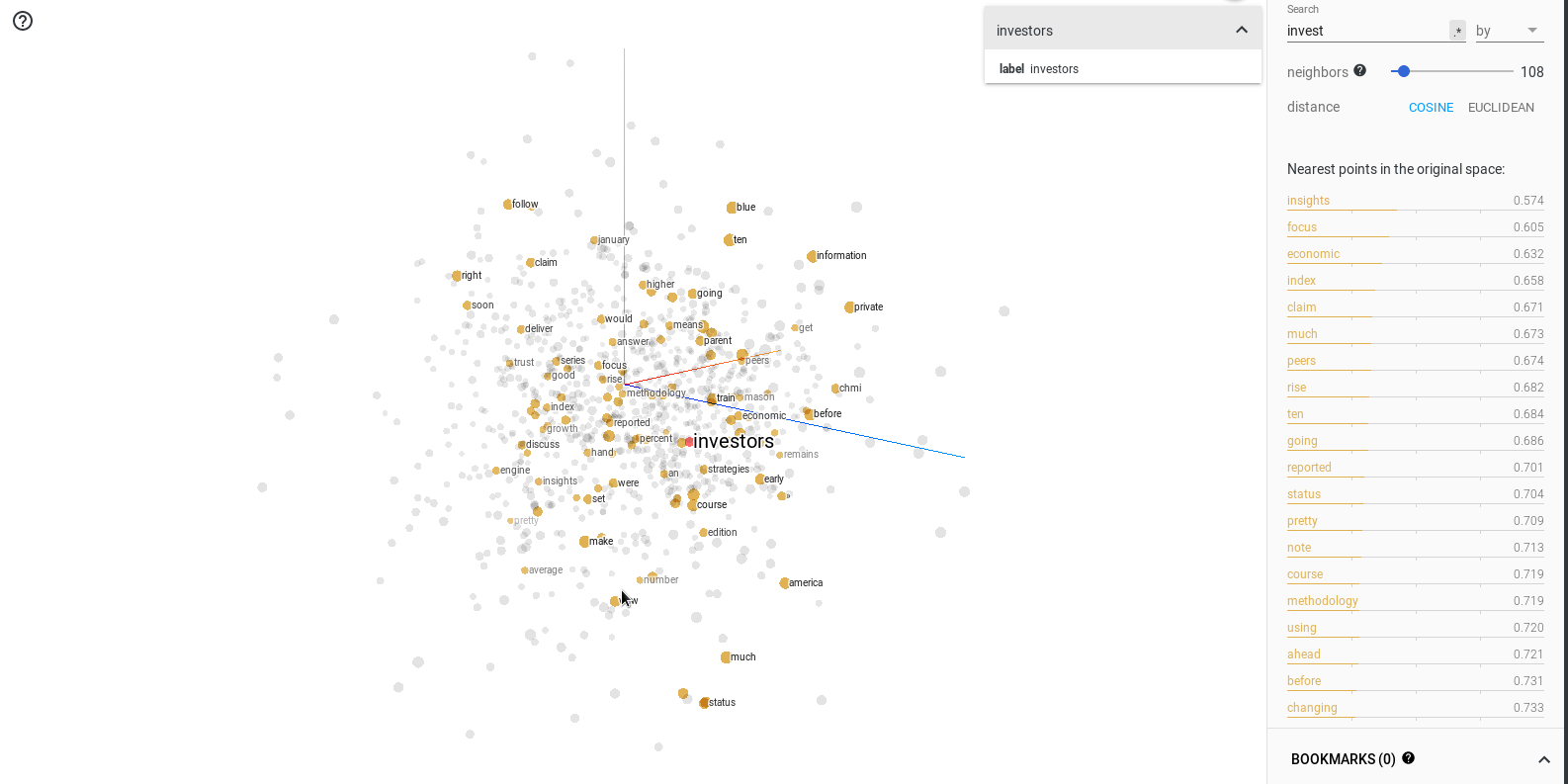
Existen además un conjunto de nodos del grafo auxiliares tal que computan diversas utilidades como la precisión de salida o la pérdida, función de diferencia entre el vector salida y el vector objetivo.

La entrenamos con nuestras bases de datos y tras un largo rato de espera conseguimos una precisión del 8% sobre nuestro vocabulario de tamaño 2000. Esta se calcula con un 20% de los datos que se reservan del entrenamiento para la validación.

Como se observa no es un valor muy alto, lo que se da debido a primero una base de datos y un tamaño de vocabulario muy reducido, incapaz por esto la red de estimar los patrones subyacentes a nuestros datos. La solución, que parece sencilla a primera vista, incrementar tanto el tamaño de la red como el tamaño de la base de datos, nos es inviable debido al factor tiempo y a la capacidad computacional, pues nuestros ordenadores son incapaces de sobrellevar la intensa computación requerida para entrenar con tan ingente cantidad de datos. 

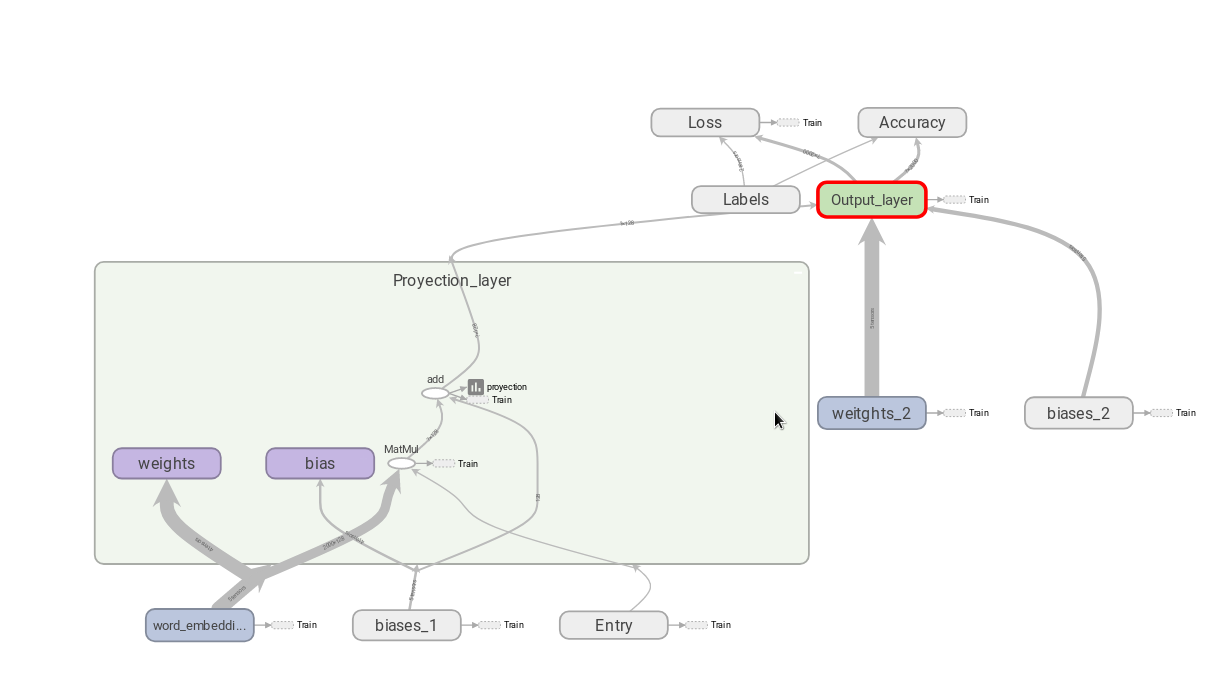
Como se observa en el gráfico de la pérdida, que cuantifica el grado de distancia entre nuestras predicciones y la realidad, esta disminuye de manera continua e incluso llega a hacerla 0 (no realmente pues no es un cálculo completo sino una estimación sobre una muestra). Esto nos indica que como hemos comentado previamente nuestra red está aprendiendo de los datos y un mayor número de éstos permitirá un mejor ajuste sobre el conjunto reservado para test.

Aun así, consideramos que dicho resultado no es totalmente insatisfactorio pues la red está adivinando la palabra correcta de un conjunto muy vasto de vocabulario. Teniendo en cuenta que el modelo aleatorio obtendrá una precisión del 1/2000 \* 100 = 0.05% nuestro modelo lo sobrepasa de manera significativa y podemos concluir que a obtenido una deseada ( aunque mejorable ) codificación.



*Proyección final del vocabulario [13]*

Guardamos por último los pesos de la matriz de proyección para poder usarlos en las etapas posteriores además de la nueva codificación de cada palabra.



*Ampliación de la capa de proyección*

**4º Etapa: Predicción del Sentimiento**

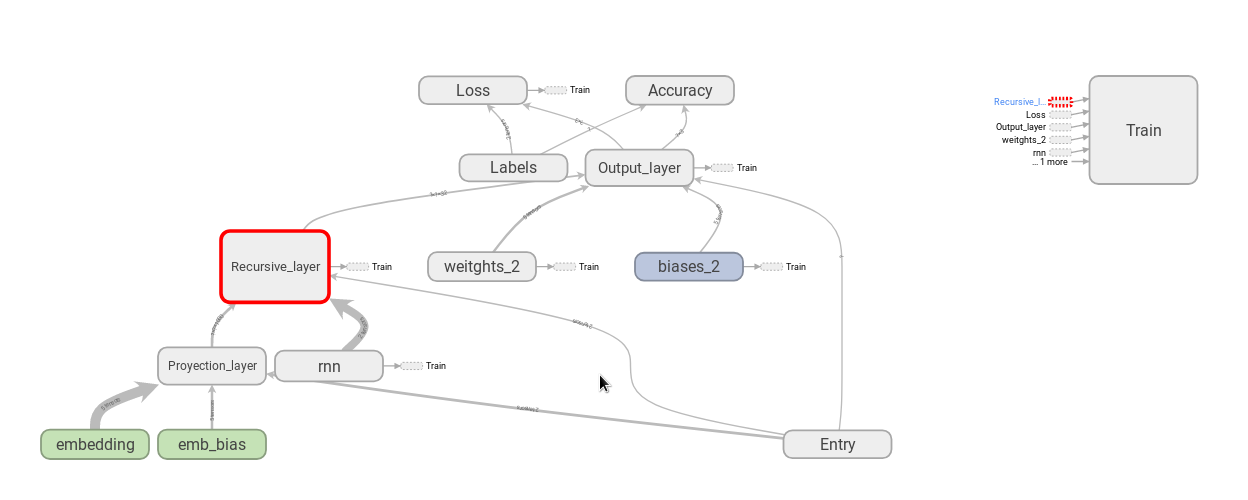
**Diseño:**

Una vez tenemos una buena representación de nuestro vocabulario, lo insertamos en nuestra segunda red neuronal para realizar, ya si, la clasificación en grupos. Estos grupos serían 0 negativo y 1 positivo.

Dado que nuestra entrada no es de tamaño fijo (número variable de palabras en cada tweet) tenemos dos opciones para nuestra segunda red neuronal:

1. Una red neuronal profunda con entrada de tamaño fijo, que requeriría un padding.
2. Una red neuronal recurrente, en particular LSTM (long-short term memory) la cual consideramos que será la adecuada, pues la literatura especializada nos indica que obtiene unos resultados más óptimos para nuestro problema particular. Además esta elimina el problema del padding pues al mantener estado interno se puede insertar palabra por palabra y recordará las entradas previas.

Se realizará, a través de nuestras bases de datos, un entrenamiento interactivo de la red neuronal. Durante la primera parte de la iteración se entrena con la base de datos etiquetada, en la segunda se procesa la base de datos no etiquetada (tweets relevantes), estos se clasifican y aquellos valores más extremos se añaden al conjunto de entrenamiento. Esto se repite hasta que todos los datos están en el conjunto de entrenamiento. Realizar esto nos permitirá tener una red neuronal mucho más robusta que si usamos solo la base de datos etiquetada y mucho más especializada a nuestro problema.



*Esquema general de la red neuronal recurrente, entrada procesada por “Proyectionn\_layer” hasta la capa de predicción “Recursive\_layer” - Obtenido con TensorBoard*

**Desarrollo:**

Como hemos comentado previamente desarrollamos esta etapa con TensorFlow, creamos una red que toma como entrada nuestro vector de datos y nuestras etiquetas.

Problema:

* Previo al desarrollo de la red, no tuvimos en cuenta las capacidades de implementación de redes neuronales recurrentes en la herramienta de nuestra elección y descubrimos que carece de la capacidad de modificar el grafo computacional de manera dinámica. Por lo que vemos necesario modificar nuestro previo diseño para tener esto en cuenta.

Solución:

* Mantener además un vector adicional que guarda la longitud de cada conjunto de entradas con la que se entrena la red, pues en TF la red neuronal recurrente requiere de estos datos, siendo imposible que acepte de manera dinámica diferentes tamaños. Por esto último es también necesario crear un relleno a cada fila de nuestro vector de entrada para que cada lote de entrada tenga siempre la misma longitud.

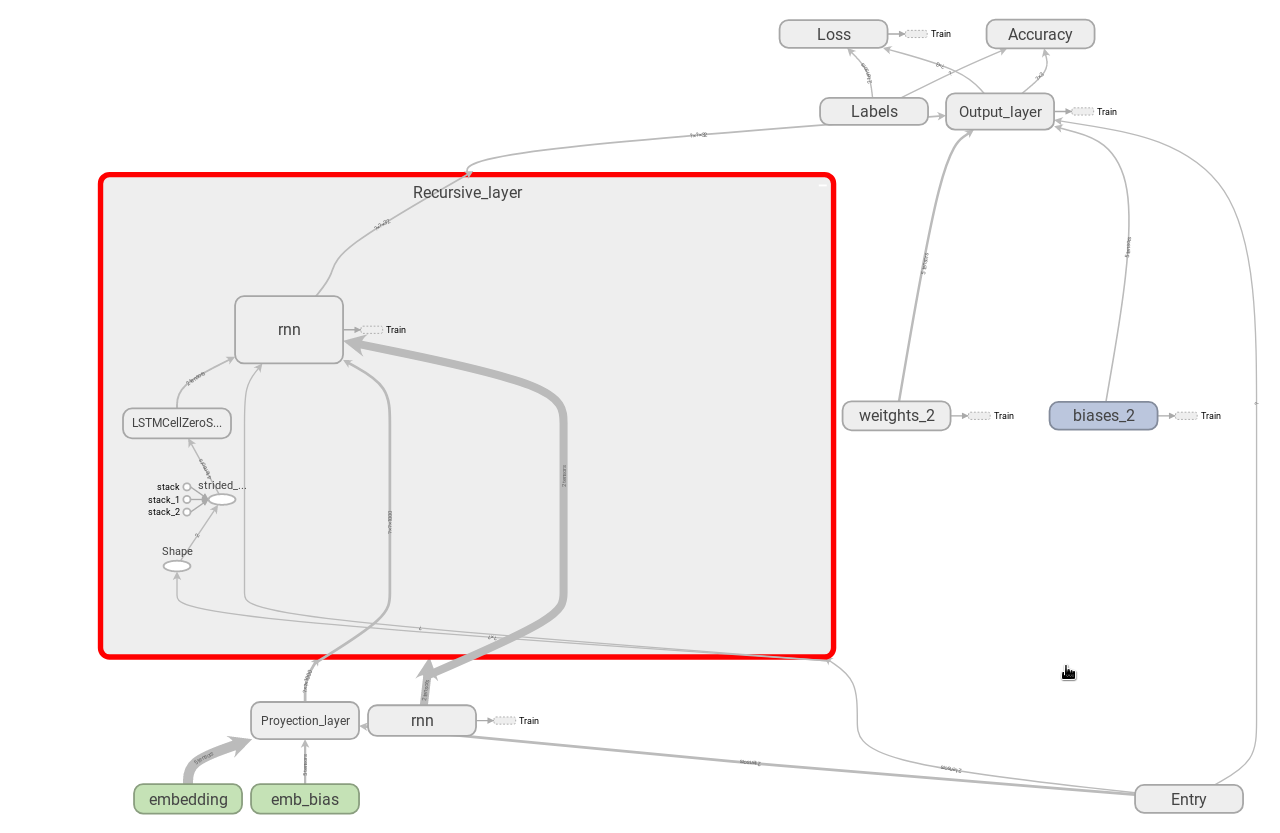
El siguiente paso es la construcción de nuestra red. Se crea el grafo computacional de TensorFlow con las operaciones *tf.multiply* para multiplicar matrices, *suma* para añadir el bias, etc. En total quedan 4 capas, la de entrada en la cual se transforma la entrada a vectores con representación “One-hot” como mencionamos previamente, la capa de proyección que comprime la información, la capa de predicción con la unidad de LSTM y la capa de salida donde se obtiene un vector de probabilidades de cada sentimiento.

Realizamos el entrenamiento híbrido con un valor de confianza de 0.8 en cualquier sentimiento, a decir, valor a partir del cual si un vector de la base de datos no etiquetada tiene este valor o más, entendido como valor extraído de nuestra red neuronal de predicción, se añade al conjunto de entrenamiento. Esto se realiza hasta que no queden más vectores de datos sin etiquetar o en la práctica hasta que se decida detener.

Una vez se realiza esto se guarda el modelo ya entrenado preparado para predecir el sentimiento de un tweet dado.

En este caso, tenemos el mismo problema que con la etapa anterior, una cantidad de datos y de cómputo muy reducida. Dado el largo tiempo que se tarda en preprocesar los datos es inviable obtener un conjunto suficientemente grande de vectores para entrenar esta red y aunque fuera viable obtener una mayor cantidad de datos de entrada, de nuevo volveríamos al problema de la capacidad de cómputo pues a mayor cantidad de datos mayor tiempo de entrenamiento.

Es por esto que la precisión de nuestra red es ínfima y no se diferencia de el modelo de clasificación aleatorio.



*Ampliación de la capa recurrente*

**5º Etapa: Estrategia de Inversión**

**Diseño:**

El paso final es utilizar todo lo que hemos obtenido para decidir si invertimos o no.

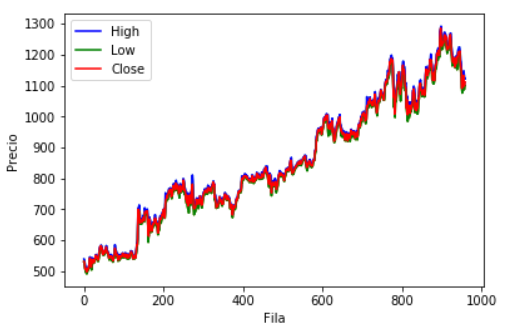
Vamos a realizar una estrategia de inversión muy simple, utilizando el histórico de precios de cada empresa (en concreto con sus precios Close, High y Low) así como su opinión pública.

En una primera instancia vamos a establecer el valor de la empresa con respecto al tiempo sin tomar en cuenta el análisis de sentimiento de los tweets para determinar cada cuanto tiempo se repite un ciclo de tendencia (por lo general suele ser de 30 días o 365 días).

Con la tendencia hayamos la predicción de los días siguientes manteniendo los últimos precios high y low constantes es decir solo predecimos los valores del precio close.

Por último aplicamos una estrategia de inversión.

**Desarrollo:**



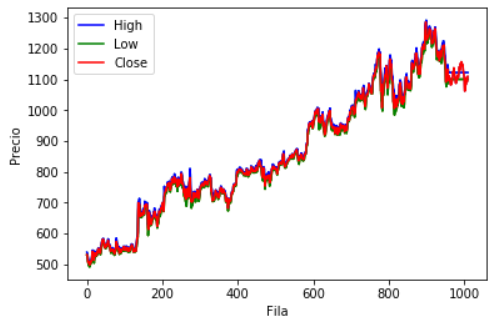
*Gráfica del histórico de precios de una empresa en particular (Google). Valor del precio de las acciones con respecto al día ordenados.*

Problema:

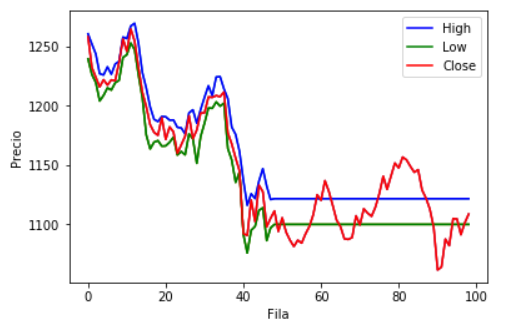
* Aunque en algunos casos tendencia es clara en otros no es tan evidente.

Solución:

* Aplicar un suavizado con periodo igual al ciclo de tendencia y los estimadores con el valor obtenido del análisis de sentimiento. Para ello implementamos las fórmulas del modelo de Winter obteniendo el suavizado de la gráfica y la predicción (de tantos días como ciclo de tendencia) producida por el análisis de sentimiento de los tweets sobre esta empresa.



*Suavizado y predicción*



*Ampliación de la gráfica anterior mostrando la predicción*

Por último aplicamos la estrategia de inversión que consiste en:

* Con la gráfica suavizada tanto los precios close, high y low, establecer como constante el high y low a partir del momento previo a la predicción del close.
* Si la predicción del precio de close supera al precio high en algún punto de la predicción futura entonces invertimos en el momento en el que empezamos a predecir.
* Si la predicción del precio de close disminuye al precio low en algún punto de la predicción futura entonces no invertimos por el momento.
* En cualquiera de los dos casos, esperamos a que llegue el día marcado por el que invertimos o no y volvemos a aplicar el algoritmo para obtener una nueva predicción a partir de ese dia.

**Comentarios finales**

Cuando empezamos el proyecto aun sabiendo que no era algo trivial y aun con la disponibilidad de varias fuentes de información la complejidad de este resultó ser más elevada de la que habíamos previsto. Caso habitual de la mayoría de los proyectos un análisis previo más profundo sobre cada iteración nos habría facilitado más las cosas previniendo de algunos errores evitables.

Más en detalle acertamos con la complejidad de la etapa de preprocesado, requiriendo al final varias iteraciones para conseguir un resultado adecuado. Aun así no previmos la complejidad de ambas redes neuronales las cuales nos han llevado, aun continuando en paralelo con el resto de las etapas, mucho más de lo previsto.

En total la duración de la realización del proyecto ha sido de poco más de 9 semanas junto con los imprevistos encontrados, revisiones, correcciones y la realización de esta memoria.

Al final, aunque sería necesario desarrollar este proyecto durante muchas más semanas para conseguir una verdadera predicción útil en la inversión, creemos que este prototipo es una prueba de concepto bastante completa que demuestra que es posible llevarlo a cabo, aunque al final no podemos validar si la inversión final sería realmente válida.

**Bases de Datos**

Yahoo Finance - <https://finance.yahoo.com/>

Twitter - [www.twitter.com](http://www.twitter.com)

MarketWatch - <https://www.marketwatch.com/>

SeekingAlpha - <https://seekingalpha.com/>

Barrons - [https://www.barrons.com/](https://www.barrons.com/?mod=BOL_LOGO)

Kaggle - <https://www.kaggle.com/kazanova/sentiment140>

**Bibliografía**

[1] Twitter Sentiment Analysis: A Review -

Kishori K. Pawar, Pukhraj P Shrishrimal, R. R. Deshmukh

[2] Efficient Estimation of Word Representations in Vector Space -  
Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean

[3] A Critical Review of Recurrent Neural Networks for Sequence Learning

Zachary C. Lipton John Berkowitz

[4] Improving Distributional Similarity with Lessons Learned from Word Embeddings -

Omer Levy Yoav Goldberg Ido Dagan

[5] A Primer on Neural Network Models for Natural Language Processing

[6] Recent Trends in Deep Learning Based Natural Language Processing

[7] Twitter Sentiment analysis and NLP

*www.towardsdatascience.com/the-real-world-as-seen-on-twitter-sentiment-analysis-part-one-5ac2d06b63fb*

Ronal Wahome

[8] Tweet analysis app

*www.csc2.ncsu.edu/faculty/healey/tweet\_viz/tweet\_app/*

[9] Renaissance Technologies

*www.rentec.com*

[10] TensorFlow

*www.tensorflow.org*

[11] Pytorch

*www.pytorch.org*

[12] Renaissance hearing before the senate permanent subcommittee on investigations

*www.hsgac.senate.gov/imo/media/doc/STMT - Renaissance (July 22 2014)2.pdf*

[13] Embeddings visualization

*www.projector.tensorflow.org*

[14] Investopedia

*www.investopedia.com*

[15] The Stanford Parser: A statistical parser

*nlp.stanford.edu/software/lex-parser.shtml*

[16] Wiktionary

*wiktionary.com*